Towards a production-ready RISC-V LLVM toolchain

Alex Bradbury, lowRISC CIC
asb@lowrisc.org @asbradbury @lowRISC

28th November 2017

LLVM/Clang vs GCC

Implementation status

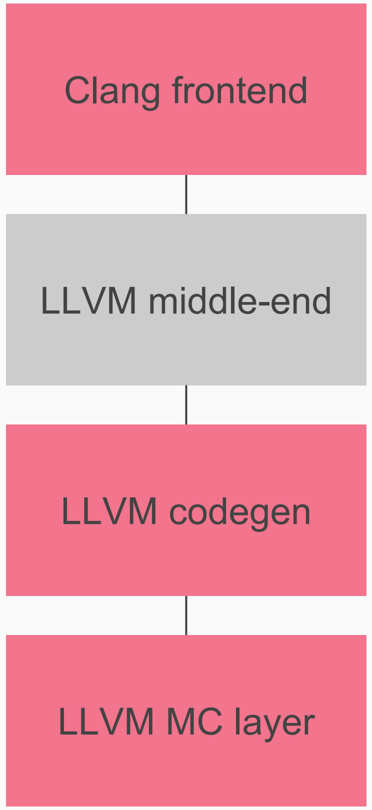
- Full time work started July 2017 thanks to funding from a RISC-V Foundation member.
- The GCC torture suite is used for initial testing
    - 1390/1390 tests compile and run for RV32I, RV32IM, RV32IMF, RV32IMFD at -O0, -O1, -O2, -O3, -Os
    - 1390/1390 tests compile and run for RV64I at -O0, -O1, -O2, -O3, -Os
- See **https://github.com/lowRISC/riscv-llvm**

Implementation status

- Moving towards larger scale tests and performance benchmarks.
  - Effort has been focused on correctness first, now moving to performance enhancements. Currently seeing +/- 20% executed insts vs GCC at O3 on computation-heavy benchmarks.
- Upstreaming of the full patchset is ongoing
- Andes Tech have started to submit an LLD (linker) port upstream and have been working on MC layer support for the C instruction set extensions
- See **https://github.com/lowRISC/riscv-llvm**

Adding RISC-V support to Clang and LLVM

- Took a “bottom-up” implementation approach
- LLVM middle-end optimisations are one of the key attractions to LLVM, and have few target-specific dependencies



Challenge:
“Bootstrapping” the effort

Challenge:
Testing the range of ISA+ABI variants

Testing: current status

- Variants:
  - RV32E, RV32I, RV64I plus combinations of the standard MAFDC extensions, plus proprietary extensions
  - ABIs: -mabi={ilp32, ilp32e, ilp32f, ilp32d, lp64, lp64f, lp64d}
  - Code models and PIC
- Approach:
  - Suite of targeted unit tests
  - GCC torture suite
  - Randomised testing via abicop (https://github.com/lowRISC/abicop)

Testing: future aims

- Want any RISC-V implementer or research group to be able to easily reach a high degree of assurance about the correctness of their toolchain modifications
- Expanded correctness and performance testing
    - SPEC and other benchmarks
    - Building a huge corpus of software (e.g. Linux userspace, FreeBSD base system)
    - Track performance over time
- Expand use of fuzz testing and automated ways of tracking quality
    - Assembler/disassembler correctness
    - Quality of debug info
    - Automated test case generation+reduction
    - Further develop abicop

Challenge:
Growing the contributor base

Growing the contributor base

- **Documentation**. See https://github.com/lowRISC/riscv-llvm
  - Support architectural innovation by reducing the difficulty of making compiler changes.
  - Incremental, continuously maintained “ideal” patch set
- Avoid duplicated effort
  - Put and end to the short term out-of-tree forks and harness the combined engineering effort of the RISC-V community.
- Growing number of external contributions:
  - LLD and MC layer support for the C extension (Andes Tech)
  - Rust port https://github.com/dvc94ch/riscv-rust-toolchain (David Craven)

Challenge:
Specifications and standardisation

Specifications and standardisation

- Initiated and authored the RISC-V Toolchain Conventions effort https://github.com/riscv/riscv-toolchain-conventions
  - Document and standardise toolchain behaviour and command line interfaces (particularly between GCC and Clang), provide preferred paths for exposing custom extensions in vendor toolchains.
  - Aim to reduce fragmentation caused by differing behaviour in GCC, Clang, and vendor-supplied toolchain distributions.
- Numerous additions to the psABI docs https://github.com/riscv/riscv-elf-psabi-doc/

Challenge: Upstreaming

Summary of approach

Put an end to the duplicated effort, allow the RISC-V community to collaborate around a common upstream toolchain without dependency on any one vendor.

- Act as a reference backend
- Extensively documented
- Clean set of incremental patches, maintained over the long term
- Upstreamed
- Contribute back, improving upstream LLVM where possible
- Grow the community, moving the RISC-V ecosystem towards collaborative development
- Invest a little extra effort at the start to get a better result over the longer term

About lowRISC

- lowRISC CIC is a UK not-for-profit (Community Interest Company, limited by guarantee) founded in 2014.
    - Serve the community of people interested in or who may benefit from open source hardware. Hobbyists, academics, startups, established companies.
    - Providing a high quality software stack is a vital complement to our work on open source hardware, novel security mechanisms, and post-design flexibility.
- Aim to bring the benefits of open source we enjoy in the software world to hardware.
- **Fulfill a role similar to Linaro in the AArch64 ecosystem, but covering reference hardware designs as well as software**

About lowRISC

- Hope to replicate the success of our RISC-V LLVM work across other areas of the RISC-V hardware/software stack.
- Spearheading targeted development efforts to enable long-term growth in the ecosystem. Sustainable and collaborative development of production-ready open source hardware/software.

Areas of interest include:

- Reference quality RISC-V core and SoC designs
- Hardware security mechanisms
- Large scale testing, benchmarking ,and verification
- Linux distribution support
- Simulation tooling and models to enable rapid architectural innovation
- Compiler toolchain and other software support

Implementation statistics

- ~20k lines added across ~185 files in LLVM projects. About 45:55 code:tests
- Work is split across 65 clean and incremental patches. 22 of these have been committed upstream so far.
- Need about ~12k lines (6k of which are tests) and ~35 patches for 100% GCC torture suite pass rate with RV32I
- Engineers from at least 7 different companies are now actively engaged or about to become actively engaged in upstream RISC-V LLVM development

What’s next

Longer term roadmap

The RISC-V LLVM ambition

Thanks

Thanks to the following people for contributing suggestions, code reviews, or patches to the RISC-V LLVM effort:

Sameer Abu Asal, Chandler Carruth, Shiva Chen, Chih-Mao Chen, Kito Cheng, David Chisnall, Simon Cook, David Craven, Hal Finkel, Eli Friedman, Ondrej Glasnak, Mandeep Singh Grang, David Kipping, James Y Knight, David Majnemer, Ed Maste, John McCall, Dylan McKay, Azharuddin Mohammed, Tim Northover, Krzysztof Parzyszek, Ana Pazos, Jordy Portman, Philip Reames, John Russo, Colin Schmidt, Pavel Šnobl, Ulrich Weigand, Eugene Zalenko, Florian Zeitz.

Overflow/backup slides

What’s next

- Performance comparisons vs RISC-V GCC
- Performance improvements
- Testing for realistic Linux applications (TLS, PIC, expand code model support)
- Merge in support for the compressed ISA (MC layer and codegen)
- Complete and promote the backend implementation documentation
- Full parity with the GNU RISC-V assembler (directives, pseudoinstructions)
- Finalise Clang ‘driver’

Longer term roadmap

- Further growth in external contributions
- Further performance and code size improvements
- Automated testing against a huge corpus (Buildroot/OpenEmbedded)
- Support for the proposed Vector extension
- Expanded performance tracking and benchmark set
- Extending the testing strategy: llvm-protobuf-mutator based assembler fuzzer, debug info quality tracking, MC layer fuzzing (equivalent of ARM's MC Hammer)
- Work with language communities such as Rust, Swift, and Julia to enable RISC-V ports using the upstream RISC-V LLVM backend.
- Further completeness: sanitizers, compiler-rt, XRay
- Instruction scheduling models for at least some of the open source RISC-V implementations
- LLDB

RISC-V LLVM ambition

We have ambitious goals for the RISC-V LLVM backend, aiming for completeness and sophistication at least on par with the AArch64/X86 backends. We hope the RISC-V LLVM backend to be the:

- Best documented
- Cleanest implementation
- Easiest to contribute to
- Highest performing and most widely used RISC-V compiler

This will:

- Lower the barrier for groups who want/need to do compiler work as part of their architectural exploration
- Support uses of RISC-V and lowRISC in education and research
- Reduce maintenance cost for those who have to maintain changes out of tree (e.g. for long term customer support)